MALLOC-OVERFLOW

An integer overflow can cause malloc() to allocate less memory than required

Sean Barnum, Cigital, Inc. [vita[1]]

2007-03-26

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 4127 bytes

Attack Category	• Malicious Input
Vulnerability Category	• Integer Overflow
Software Context	• Memory Management
Location	
Description	An integer overflow can cause malloc() to allocate less memory than required. This is an integer overflow problem, not an issue with the malloc() function. This problem occurs if the input to malloc() contains integer arithmetic that causes an overflow. For example, the function call malloc(0x4000001 * sizeof(int)) would only allocate 4 bytes on a platform where sizeof(int) = 4. This is because the input to malloc is a 32-bit value, which can only hold values up to 0xFFFFFFFF, and 0x40000001 * 4 = 0x100000004. Since the function call returns normally with a valid pointer, the program may unknowingly overwrite other data that exists past the allocated 4 bytes.
APIs	**Function Name** / **Comments** malloc xmalloc calloc GlobalAlloc
Method of Attack	The exploit can occur when the attacker controls the input to malloc(). He can cause it to allocate less than the number of bytes expected. For example, if an array size is calculated based on unchecked user input, the attacker can cause the program to inadvertently overwrite other data or display potentially sensitive information stored past the allocated memory.
Exception Criteria	When the input to malloc() is a constant provided by the programmer.

1. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/35-BSI.html (Barnum, Sean)

<table>
<tr><td>Solutions</td><td>
<table>
<tr><th>Solution Applicability</th><th>Solution Description</th><th>Solution Efficacy</th></tr>
<tr><td>Whenever a dynamic calculation is done to determine the amount of memory to be allocated.</td><td>C does not provide any integer overflow detection or handling. The only (efficient) way to prevent this error from occurring is to perform bounds checking on all operands in any calculations performed to generate the input to malloc().</td><td>Effective if checks are appropriate and correctly implemented.</td></tr>
</table>
</td></tr>
<tr><td>Signature Details</td><td>malloc(int * int)
malloc(int + int)
... or any math operator ...</td></tr>
<tr><td>Examples of Incorrect Code</td><td><pre>[...]
int array_size;
[...]
malloc(array_size * sizeof(int));
[...]</pre></td></tr>
<tr><td>Examples of Corrected Code</td><td><pre>[...]
int array_size;
[...]
if (array_size > UINT_MAX /
sizeof(int)) return ERROR;
malloc(array_size * sizeof(int));
[...]</pre></td></tr>
<tr><td>Source Reference</td><td>• http://www.secureprogramming.com/?action=view&feature=recipes&recipeid=14[2]</td></tr>
<tr><td>Recommended Resource</td><td></td></tr>
<tr><td>Discriminant Set</td><td>
<table>
<tr><td>Operating System</td><td>• Windows</td></tr>
<tr><td>Languages</td><td>• C
• C++</td></tr>
</table>
</td></tr>
</table>

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com